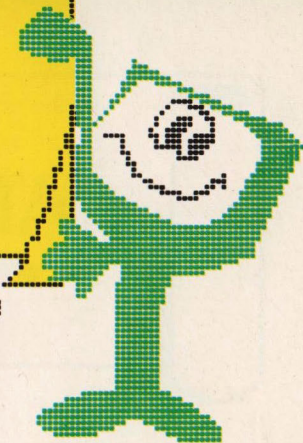
VIDEO BASIC

20 VIDEOLEZIONI DI BASIC
PER IMPARARE CON LO SPECTRUM

I computer del futuro
L'intelligenza artificiale

Il BASIC e la memorizzazione
dei programmi

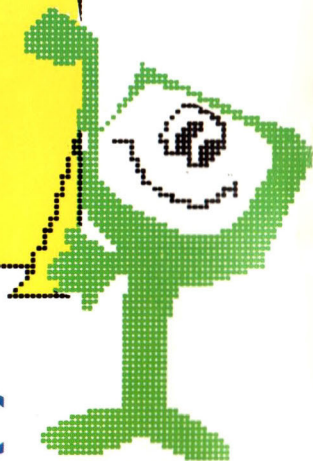
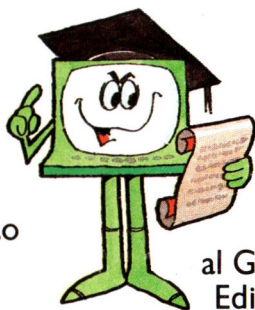
Gli interrupt

Usare la ROM

Videoesercizi

Videogioco n° 20

20 Spectrum

16K/48K/PLUS

VIDEO BASIC TI ASPETTA IN EDICOLA IL 2 DICEMBRE CON UN NUMERO SPECIALE A SOLE L. 2.000

Completa il corso di "Video Basic" con questo fascicolo che contiene l'indice dell'opera, per una veloce e comoda consultazione, per un facile ripasso.
In più troverai una gradita sorpresa!
Un test finale da spedire al Gruppo Editoriale Jackson per mettere alla prova le tue capacità.

VIDEO BASIC SPECTRUM
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea - Via Indipendenza 88 - Como
Redazione software:
Francesco Franceschini, Roberto Rossi,
Alberto Parodi, Luca Valnegri
Segretaria di Redazione:
Marta Menegardo
Progetto grafico:
Studio Nuovaidea - Via Longhi 16 - Milano
Impaginazione:
Silvana Corbelli
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.r.l. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

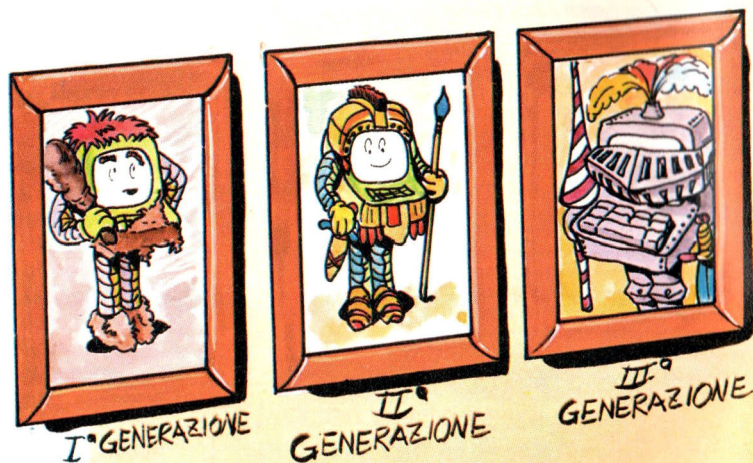
Gruppo Editoriale Jackson

SOMMARIO

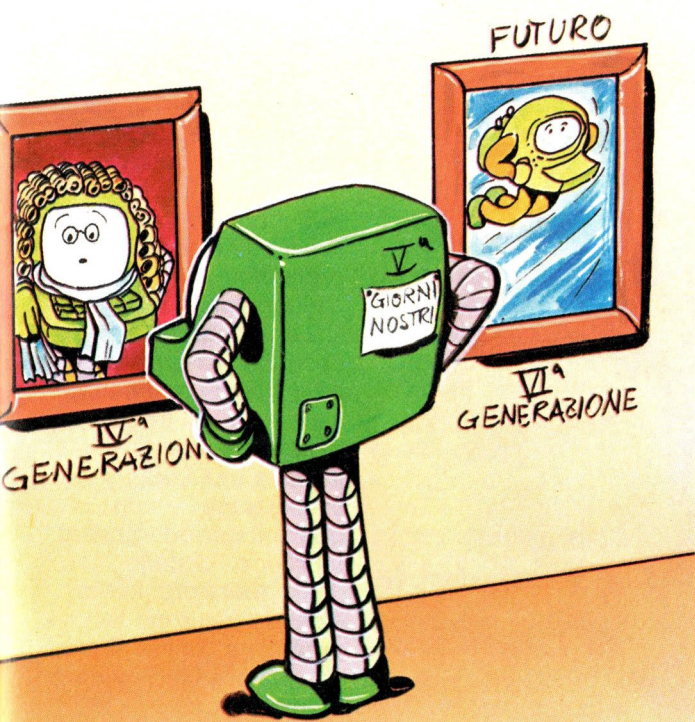
Introduzione

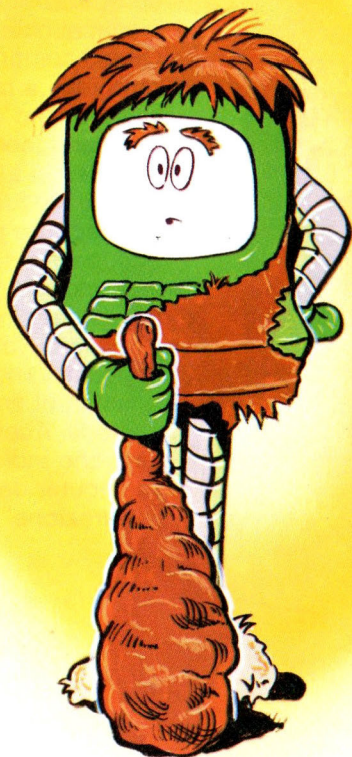
Il tuo Spectrum ha qualità e pregi impensabili sino a pochi anni fa; quello che i laboratori di ricerca ci stanno preparando, però, va ben oltre: computer ultraveloci, capaci di elaborare più informazioni in parallelo, ma soprattutto computer "intelligenti". La fantascienza, insomma, non è più tanto futuribile.
Per restare, comunque, coi piedi nel presente dobbiamo perfezionare quanto appreso e apprendere del nuovo.
Ecco allora come risparmiare tempo e memoria, i puntatori, gli interrupt, e ancora linguaggio macchina.
Morale. Se il computer non ragiona e parla ancora come un uomo, occorre saper ragionare e parlare come il computer.

I computer del futuro

Per quanto molte persone ritengano tuttora il contrario, i computer non lavorano assolutamente per magia: ormai sappiamo benissimo che tutti i risultati ottenibili mediante un elaboratore elettronico non sono altro che il prodotto di un complesso e velocissimo insieme di operazioni elementari, svolte all'interno dei circuiti e delle memorie della macchina. Ciò che ad alcuni può ancora sembrare fantastico, è in realtà un preciso e logico sviluppo di un settore tecnologico che poggia su solide e rigorose basi teoriche e scientifiche.
Soltanto fino a pochi decenni fa nessuno avrebbe tuttavia potuto pronosticare la nascita e soprattutto lo sviluppo di una tecnologia così rivoluzionaria come quella elettronica: non esisteva infatti alcun presupposto che lasciasse intravvedere un avvenire così carico di sviluppi. Adesso che viviamo in pieno nella cosiddetta "era

elettronica" abbiamo invece sufficienti "orizzonti" per poter immaginare quali saranno i probabili progressi nel settore dei computer.
Le strade che in questo momento vengono battute dagli studiosi di tutto il mondo meritano comunque di essere descritte e analizzate.

I calcolatori del passato

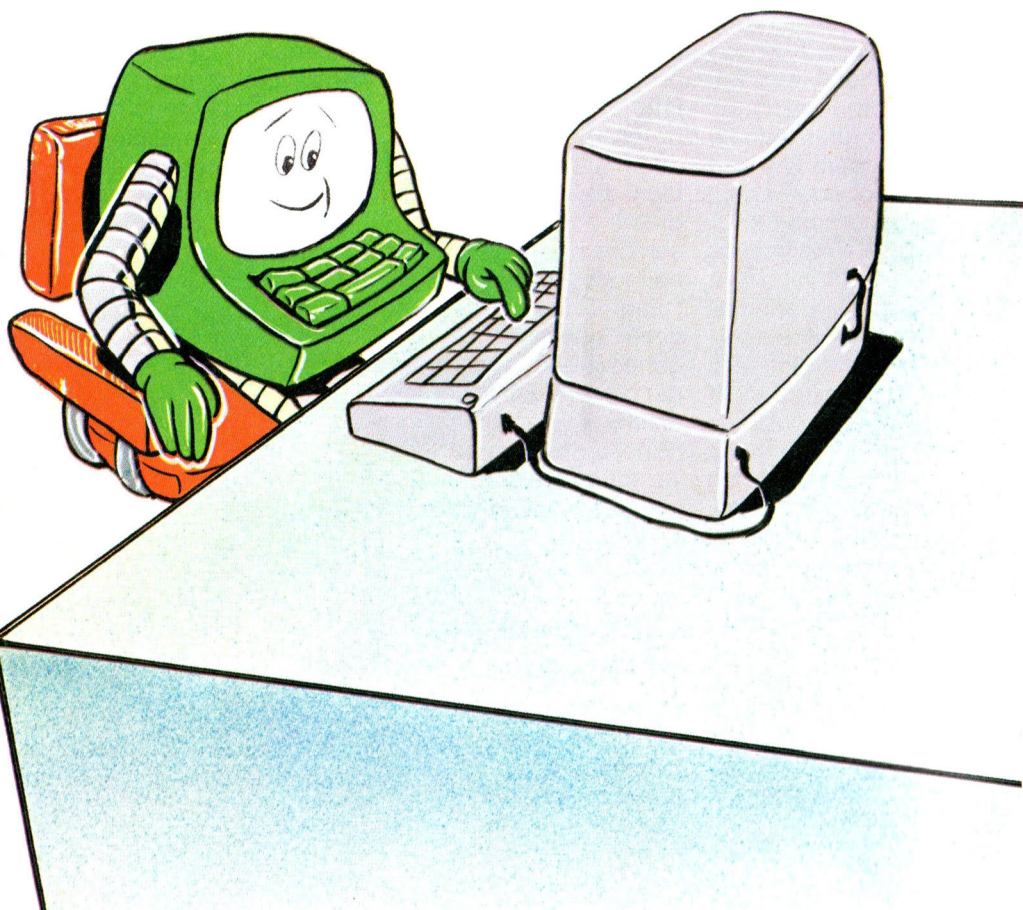
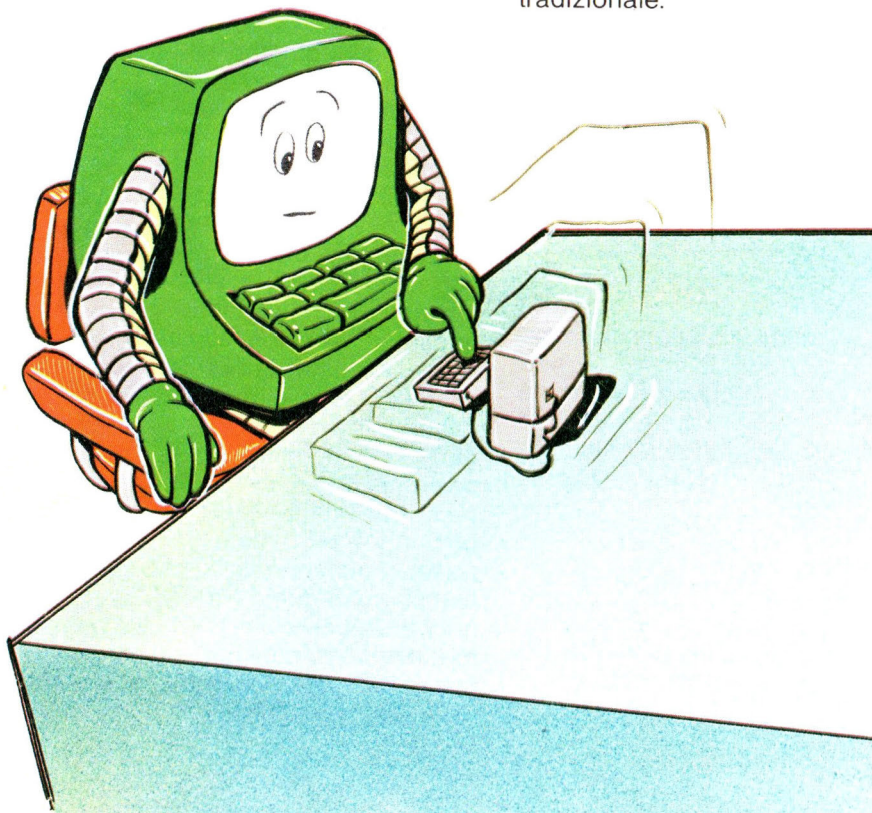
Prima di pensare al futuro diamo innanzitutto un'occhiata al passato: il detto "preparati al futuro guardando nel passato" è più che mai appropriato per chi, come noi, desidera fondare in maniera concreta le proprie ipotesi. Trascurando gli studi e le teorie logiche che hanno consentito la nascita e l'evoluzione del calcolo automatico, andiamo all'inizio degli anni '50, quando ancora nel campo elettronico imperavano le valvole e i diodi. La guerra era finita da poco e i calcolatori erano appena agli albori: in quel periodo cominciarono comunque ad apparire le prime macchine elettroniche capaci di eseguire automaticamente calcoli ed operazioni matematiche.
Questi computer raffrontati a quelli di oggi sembravano dei dinosauri, con dimensioni a dir poco enormi (l'equivalente di un moderno personal occupava un intero laboratorio), composti da chilometri di cavi e migliaia di valvole. In più consumavano notevoli quantità di energia elettrica. Adesso vengono indicati come "computer della prima generazione".

HARDWARE

Le macchine di questo tipo erano principalmente a disposizione dei grossi centri di ricerca militari od universitari e di grandi industrie: l'hardware era infatti troppo ingombrante e soggetto a guasti per garantire immediati sviluppi o applicazioni commerciali. Anche per il software le cose non andavano molto meglio: linguaggio macchina e linguaggi simbolici elementari di tipo assemblatore ponevano grossi limiti di utilizzo.
Però, pian piano gli elaboratori cominciarono comunque a diffondersi: occorreva però renderli più affidabili e meno ingombranti. Il fatto decisivo fu la scoperta del transistor. In un computer un transistor assolve la stessa funzione di una valvola: il transistor consuma però molto meno energia delle valvole, è di dimensioni molto minori e soprattutto dispone di una vita media di gran lunga superiore. Inoltre, - fatto anche questo essenziale - costano meno.
Benché sostanzialmente simili agli elaboratori precedenti sotto l'aspetto della logica, questi nuovi sistemi (detti della seconda generazione) se ne distaccavano per vari aspetti, primo tra tutti il dimensionamento.
Iniziava una vera "rivoluzione" e con quella il periodo di vero e proprio "decollo" dell'elaboratore. Molte aziende capirono la praticità e l'utilità di un calcolatore elettronico e lo installarono

richiedendo nello stesso tempo una maggiore potenza e ancora più elevata velocità di elaborazione. Nuove possibilità di elaborazione apparvero con l'introduzione delle memorie a nucleo magnetico, allargando ulteriormente le applicazioni e i possibili sviluppi.
Il più grosso passo avanti venne comunque fatto con la creazione dei circuiti integrati o chip: questa nuova tecnologia generò gli elaboratori della "terza generazione". I circuiti integrati introdussero nuovi miglioramenti, miniaturizzando e raffinando i componenti della seconda generazione. Essi inoltre costavano ancora meno delle "vecchie" piastre a transistor.
Contemporaneamente agli sviluppi dell'hardware, anche la programmazione aveva fatto passi da gigante, proponendo in continuazione nuove tecniche e applicazioni, grazie a linguaggi sempre più potenti e nello stesso tempo più "umani". I computer della "quarta generazione" sono quelli dei giorni nostri: piccoli, efficienti, affidabili, ma - nonostante le apparenze - ancora ulteriormente migliorabili. Vediamo in quali modi.

Uno sguardo al futuro

La ricerca scientifica è già a uno stadio avanzato: si stanno infatti battendo molte strade, sia "hardware" che "software". Le principali aree di studio dal punto di vista costruttivo riguardano soprattutto la superconduttività e l'elaborazione parallela, mentre l'informatica vera e propria punta tutte le proprie speranze verso quel settore di indagine, estremamente stimolante, che prende il nome di "intelligenza artificiale". Al solito, lo scopo è quello di realizzare elaboratori sempre migliori, sempre più versatili, sempre più utili; che lavorino più in fretta, memorizzino più informazioni, richiedano meno potenza, occupino meno spazio e costino sempre meno.

I computer a superconduttori

In realtà, i computer a superconduttori esistono già da qualche tempo: tuttavia le possibilità di sviluppo che essi sembrano in grado di offrire li pongono in un settore che appartiene più al domani che all'oggi.
I computer attualmente costruiti hanno raggiunto velocità di elaborazione talmente elevate da poter essere confrontate alle velocità con cui gli elettroni si muovono nei circuiti elettronici. È chiaro che se gli spostamenti degli elettroni all'interno dell'elaboratore (ricordiamoci che il flusso degli elettroni all'interno dei circuiti costituisce la corrente elettrica) sono più lenti delle possibilità di calcolo dell'unità centrale, il tempo di esecuzione è costretto a subire un rallentamento. In altre parole, le informazioni impiegano un certo tempo (per quanto piccolissimo) per

andare da un punto all'altro dei circuiti elettronici: se questo tempo è superiore a quello della velocità di calcolo della CPU, questa è per forza di cose costretta a "rallentare", con ovvie conseguenze sulla velocità di lavoro dell'intero sistema. Questo problema, a prima vista apparentemente insolubile (non esiste infatti alcuna possibilità di "accelerare" il moto degli elettroni nei conduttori, essendo quest'ultimo una caratteristica specifica dei materiali), viene allora risolto ricorrendo a particolari leghe di metalli conduttori, che godono della singolare proprietà di opporre - a temperature estremamente basse (più di 100 gradi al di sotto dello zero!) - una resistenza al moto degli elettroni di gran lunga più bassa di quella che offrono alle normali temperature ambientali. Il problema viene quindi brillantemente risolto: le velocità di elaborazione dei computer a superconduttori (superconduttività è il nome del processo fisico che abbiamo appena visto) raggiungono infatti valori assolutamente impensabili nei comuni computer a tecnologia tradizionale.

I computer paralleli

L'idea che sta alla base della tecnica dei computer paralleli è di una semplicità quasi disarmante: anziché usare una sola unità centrale, nei computer paralleli si utilizzano più unità centrali, che lavorano in modo simultaneo o, come si usa dire più comunemente, in parallelo. Le possibilità che si propongono sono estremamente allettanti: teoricamente basta aggiungere altre CPU e le potenzialità di elaborazione di qualsiasi computer diventano praticamente senza limiti.

Naturalmente, come nella maggior parte dei progetti che a parole sembrano "semplici", gli spazi di azione che separano il dire dal fare appaiono estremamente impegnativi.
I problemi principali dei computer paralleli non risiedono infatti unicamente nelle pure e semplici disposizioni circuitali (tutt'altro che facili da risolvere), ma anche (e soprattutto) nelle modalità di programmazione che queste macchine richiedono.
Occorre infatti disporre di un linguaggio che riesca a "sincronizzare" le operazioni di tutte le CPU, evitando interferenze ed accavallamenti reciproci, con le ovvie (e deleterie) conseguenze che ne potrebbero derivare. A tutt'oggi un linguaggio di questo genere non esiste: i tentativi finora sperimentati lasciano comunque intravvedere ben più di semplici speranze. In tempi recentissimi sono state comunque prodotte e poste in commercio macchine di questo tipo, anche se, per il momento non possono essere sfruttate al pieno delle loro possibilità.

Intelligenza artificiale

L'intelligenza artificiale è senza alcun dubbio il settore che nello sviluppo della scienza dei computer avrà la maggiore influenza sul nostro modo di vivere nei prossimi decenni.
Per generazioni gli scrittori di fantascienza hanno pronosticato l'evoluzione di macchine più o meno intelligenti, capaci di assolvere molte delle funzioni eseguibili soltanto dagli esseri umani. Con ogni probabilità le vicende di androidi e umanoidi resteranno però di esclusivo dominio della lettura fantastica e avveniristica.
Al giorno d'oggi si pensa piuttosto alle macchine "intelligenti" come a sistemi capaci di prendere determinate decisioni al momento più opportuno.
L'esatta definizione dell'intelligenza di una macchina è un argomento in continua evoluzione: per quanto gli esperti si accaniscano in continui dibattiti su questo tema, si può comunque accettare come definizione standard

quella che venne proposta nei "lontani" anni '40 da un vero e proprio pioniere dell'informatica: Alan Turing. Piuttosto che elencare una serie di criteri da soddisfare per classificare un computer come intelligente, egli si limitò a dare una opinione ben più pratica del problema.
Turing affermò che se una persona non è in grado di distinguere se certe risposte le arrivano da una macchina o da un'altra persona, allora la macchina che ha eventualmente elaborato quelle risposte è da classificarsi come intelligente. Tale prova costituisce la base del famoso "Test di Turing", nel quale un operatore umano deve sostenere - attraverso una tastiera e un terminale - una certa conversazione, cercando di costringere l'interlocutore a svelare la propria identità di uomo o di macchina.
I centri di ricerca di tutto il mondo stanno portando avanti studi e indagini sulla intelligenza artificiale, e sembra che anche in questo caso sia solo questione di anni per arrivare a risultati effettivi. Il Giappone ha già anticipato tutti, annunciando che il suo elaboratore della "quinta generazione" vedrà la luce al massimo entro il 1995.
L'intelligenza artificiale ha già fatto il suo ingresso in molti settori, come per esempio quello dei "sistemi esperti": con questo nome si intendono quegli elaboratori specializzati in grado di eseguire un certo compito altrettanto bene (e in alcuni casi anche meglio) degli esperti umani. Un tipico esempio di utilizzo di questa tecnologia lo possiamo vedere tutti i giorni guardando le previsioni del tempo, elaborate quotidianamente appunto mediante l'ausilio di sistemi esperti. Anche nel campo delle diagnosi mediche sembra si stiano facendo passi da gigante: l'ipotesi del computer-dottore non è quindi troppo azzardata.
La maggior parte degli studi sull'intelligenza artificiale vengono condotti utilizzando particolari linguaggi, creati appositamente per questo scopo (in genere LISP e PROLOG). Poiché questi ultimi richiedono potenze di calcolo ben superiori a quelle offerte dai piccoli computer, nelle nostre case l'intelligenza artificiale entrerà tra pochi anni grazie alla telematica che consentirà di collegare l'home computer ad un grande sistema intelligente.
Ciò non significa comunque che il campo non sia affrontabile - anche se a livello hobbystico - con un normale personal computer.

Risparmiare tempo e memoria

Nella programmazione, come in molti altri settori del lavoro umano, è stata costruita una scala di valori, che classifica le varie tecniche in "buono", "non molto buono", "pessimo". Quando hai cominciato a usare il tuo Spectrum, essere in grado di scrivere un programma funzionante era già di per sé un valido risultato. Adesso che sei diventato molto più padrone della situazione, e disponi di conoscenze e capacità che inizialmente non avevi, è giunto il momento di affrontare un discorso che - a prima vista - ti potrebbe sembrare poco rilevante, ma che in realtà è importantissimo per migliorare la tua tecnica di programmazione.

Vogliamo infatti vedere come migliorare ed aumentare la velocità di esecuzione dei tuoi programmi BASIC, senza per questo influenzarne la qualità, la leggibilità e l'occupazione di memoria. Non sempre velocità di esecuzione e risparmio di memoria sono conciliabili: esiste comunque un certo numero di "trucchetti", che di volta in volta possono tornare utili. Vediamone quindi alcuni:

- **Istruzioni multiple.**
Il porre più istruzioni in una stessa linea aiuta a minimizzare la lunghezza del programma, risparmiando sui numeri di linea e di conseguenza sull'occupazione di memoria. Lo svantaggio e la limitazione principale di questa tecnica risiede però nella scarsa leggibilità e nella difficile modificabilità delle varie righe. È quindi meglio non abusare troppo con questa pratica.

- **Variabili.**
Le variabili dovrebbero essere chiamate con i nomi più brevi possibili. Questo aiuta a risparmiare memoria ed accelera il lavoro dell'interprete BASIC. Anche le costanti (sia numeriche che alfanumeriche) - se utilizzate di frequente - conviene assegnarle a delle variabili. Per esempio, anziché scrivere:

```
10 POKE 15143,12:POKE 15143,70:POKE 15143,20
```

conviene fare:

```
10 LET A=15143:POKE A,12:POKE A,70:POKE A,20
```

LINGUAGGIO

L'interprete BASIC dovrà in questo modo convertire una sola volta il numero 15143 in un formato comprensibile alla CPU, riducendo oltretutto anche lo spazio occupato in memoria.

- **REM.**

Bisogna limitare al massimo l'uso dei commenti nei programmi. Questa indicazione sembra in apparente contraddizione con quanto avevamo detto finora, e cioè che la documentazione dei programmi dovrebbe essere la più abbondante possibile. I programmatori più esperti risolvono allora il problema conservando due copie dello stesso programma: la prima, commentata in tutti i suoi aspetti, serve per capire il funzionamento del programma e per apportarvi eventuali modifiche; la seconda, che dovrà girare nel computer, viene privata di qualsiasi commento, in modo da evitare "inutili" sprechi di memoria.

• GOTO.
Ogni volta che l'interprete deve eseguire un salto la sequenza delle istruzioni subisce un improvviso cambiamento, spostandosi a un altro punto del programma. L'entità di questo cambiamento viene definita specificando nell'istruzione GOTO il numero di linea a cui saltare. L'interprete BASIC non dispone però di tecniche di ricerca particolari per individuare tale numero di linea ed esegue quindi una lenta ricerca sequenziale a partire dall'inizio del programma. Il tempo necessario per eseguire una istruzione di salto dipende dunque dalla lunghezza del testo e dalla distanza della linea indicata nel GOTO dall'inizio del testo. Quando si desidera accelerare al massimo la velocità di esecuzione è pertanto necessario valutare attentamente questo fatto, cercando di limitare al massimo la presenza di GOTO (avvantaggiando anche la leggibilità del programma) o - se proprio non se ne può fare a meno - avvicinando al massimo le istruzioni a cui saltare all'inizio del programma.

• GOSUB.
L'utilizzo delle subroutine consente un notevole risparmio di memoria, dal momento che permette di evitare la ripetuta scrittura di gruppi di istruzioni. Per i comandi GOSUB valgono tuttavia le stesse considerazioni fatte per i GOTO; la cosa migliore (e questo è di semplice applicazione) è allora quella di porre tutte le subroutine - all'inizio dei programmi, anziché alla fine come siamo abituati - dando la precedenza (cioè scrivendole prima) a quelle che il programma utilizzerà più frequentemente.
La raccomandazione più importante è comunque sempre la stessa e cioè che qualsiasi programma può diventare notevolmente più corto e veloce, se viene strutturato con una buona logica.

Come lavora il BASIC: i puntatori

Un puntatore è una porzione della memoria del computer (di solito molto piccola: una o due locazioni) il cui contenuto è costituito da indirizzi utili al funzionamento del sistema. Ci spieghiamo meglio con un esempio. Quando accendi il tuo Spectrum, l'interprete BASIC deve mettersi immediatamente nella condizione di accettare le linee di programma che vorresti battere. Per fare questo è chiaramente necessario che l'interprete conosca la zona di memoria in cui dovrà immagazzinare le varie istruzioni; l'indirizzo di tale zona sarà allora contenuto in un apposito puntatore, letto dall'interprete durante la routine di accensione (o inizializzazione) del sistema. In ogni computer esistono numerosi puntatori, ciascuno dei quali con una specifica funzione. La loro principale utilità risiede nel fatto che grazie ad essi è possibile riferirsi con

minimo sforzo a particolari aree della memoria, consentendo quindi, con estrema facilità, eventuali modifiche o aggiornamenti. Tra breve scopriremo che anche la memorizzazione delle linee di programma ricorre ampiamente a questa tecnica.

La memorizzazione dei programmi

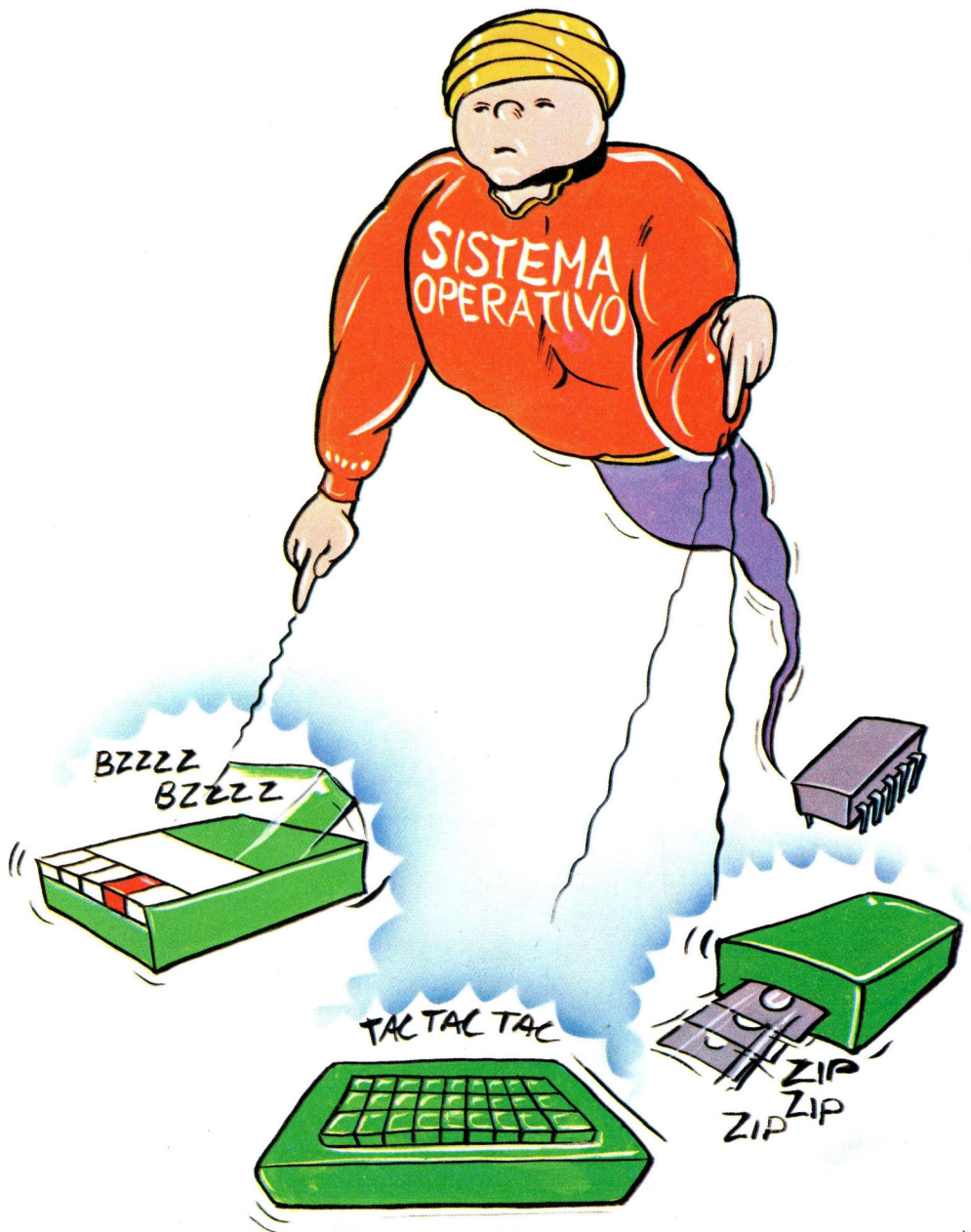
Quando batti in BASIC una qualsiasi riga di programma le linee vengono memorizzate in questo modo:

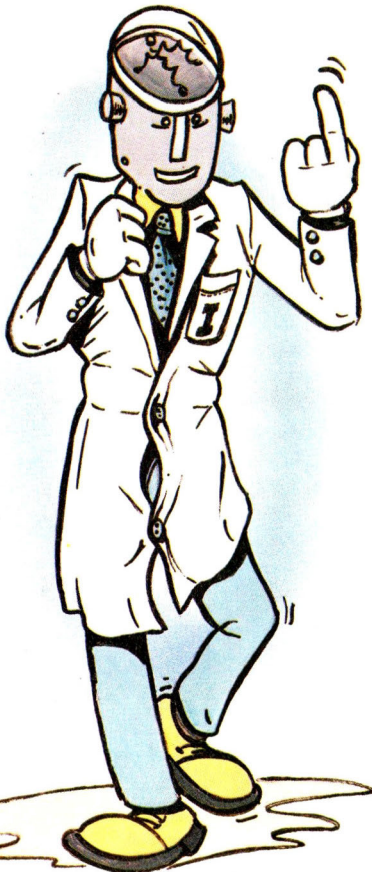
- 2 byte per il numero di linea utilizzati in modo che il byte più significativo preceda il byte meno significativo.

- 2 byte come contatori della lunghezza della linea, dal primo carattere del testo alla fine della riga (compreso il carattere ENTER). In questo caso i due byte sono utilizzati nel modo byte basso-byte alto.

- Il testo delle parole BASIC scritte in codice ASCII, dove:
— le parole e i simboli riservati occupano un solo byte. Per risparmiare memoria le parole riservate vengono infatti convertite in particolari codici numerici, chiamati TOKEN. Così, per esempio, la parola GOTO non viene memorizzata con 4 lettere separate ("G", "O", "T", "O") e quindi con 4 byte, ma con l'unico codice 236;
— i parametri e i segni di punteggiatura sono presentati carattere per carattere in codice ASCII;
— i parametri numerici sono rappresentati in due forme diverse: come stringhe di numeri e come numeri in formato esponenziale. La stringa viene usata durante il listing (cioè quando desideri leggere ciò che hai battuto), mentre il formato esponenziale viene usato durante l'esecuzione del programma (è questo un modo estremamente ingegnoso per risparmiare all'interprete la fatica di convertire in continuazione i vari numeri). Le due forme vengono usate separate da un codice numerico (14), in modo che l'interprete sappia sempre quale scegliere.

- Il codice 13 per ENTER.
Per segnalare la fine del programma non viene usato alcun codice particolare. L'interprete segnala semplicemente se il byte successivo al codice ENTER ha i primi due bit di sinistra a 01, 10 o 11: se ciò accade, sicuramente non si tratta di una nuova riga di programma.
Segue un programma che illustra quanto appena visto. Esso infatti "legge" se stesso nella memoria dello Spectrum.

SISTEMA
OPERATIVO
BZZZZ
BZZZZ
TAC TAC TAC
ZIP
ZIP
ZIP

```
 1 REM prova
10 LET p=PEEK 23635+256
   *PEEK 23636
15 LET v=PEEK 23627+256
   *PEEK 23628
20 LPRINT "p=";p,"v=";v:LPRINT
25 FOR k=p TO v
30 LET x=PEEK k
35 LPRINT x;" ";:IF x=13 THEN LPRINT
   :LPRINT
40 NEXT k
45 STOP
```

Vediamo brevemente il significato delle varie istruzioni:

linea 10: calcola il puntatore p di inizio programma;

linea 15: calcola il puntatore v di inizio variabili;

linea 20: stampa il valore di p e v;

linee 25-40: è un ciclo che stampa il contenuto dei byte da p a v, andando a capo quando un byte contiene 13 (che è il codice di ENTER);
linea 45: fine del programma.
Eseguendolo, otterrai in uscita una cosa del genere:

```
p=23755          v=23935

0 1 7 0 234 112 114 111 118 97 13

0 10 39 0 241 112 61 190 50 51 54
50 55 14 0 0 75 92 0 43 50 53
54 14 0 0 0 1 0 42 190 50 51 54
50 56 14 0 0 76 92 0 13
..............
```

e così via.

Cerchiamo adesso di capire cosa significano queste serie di numeri. Il programma occupa 23935−23755=180 byte: questa è infatti la memoria totale impegnata dalle istruzioni.
Ciascun gruppo di valori è la rappresentazione

numerica con cui il tuo Spectrum ha memorizzato le istruzioni. Per esempio, in corrispondenza di 1 REM prova troviamo:
0 1, che è il numero di linea : 0*256+1=1

7 0, 0*256+7=7 è la lunghezza del testo dell'istruzione + ENTER

234 è il codice ASCII di REM

112 114 111 118 97, sono i codici ASCII dei caratteri della parola "prova", scritta in minuscolo

13, codice di ENTER.

Per quanto riguarda invece la riga

```
10 LET p=PEEK 23635+256*PEEK 23636
```

troviamo:
0 10, numero di linea 0*256+10=10

39 0, 0*256+39=39 è la lunghezza del testo + ENTER

241, è il codice ASCII di LET

112, è il codice ASCII di p minuscolo

61, è il codice ASCII di =

190, è il codice ASCII di PEEK

50 51 54 51 53, sono i codici ASCII delle cifre del numero 23635 (utilizzati quando si fa il listing)

14, segnalatore inizio numero in formato esponenziale (utilizzato dal computer durante l'esecuzione)

0 0 83 92 0, sono i 5 byte della rappresentazione del numero intero 23635

43, è il codice ASCII di +

50 53 54, codici ASCII delle cifre del numero 256

14, segnalatore inizio numero in formato esponenziale

0 0 0 1 0, sono i 5 byte della rappresentazione del numero intero 256

42, codice ASCII di *

190, è il codice ASCII di PEEK

50 51 54 51 54, sono i codici ASCII delle cifre del numero 23636

14, segnalatore inizio numero in formato esponenziale

0 0 84 92 0, sono i 5 byte della rappresentazione esponenziale del numero intero 23636.

Le altre linee del programma possono essere analizzate nel

modo appena visto: come esercizio lo lasciamo fare a te. Ti proponiamo invece un altro interessante programma, il cui compito è quello di stampare tutte le parole riservate che in BASIC vengono "tokenizzate", cioè ridotte a un solo codice numerico.
Nella memoria ROM dello Spectrum, dal byte di indirizzo 150 al byte di indirizzo 516, sono infatti contenute le descrizioni dei vari token utilizzati dal programma.

```
10 REM token
12 LPRINT “Dec. Esadec.”
14 LPRINT “ASCII ASCII TOKEN”:LPRINT
16 LET n=165:LET sw=0
18 LET h$=“0123456789ABCDEF”
20 FOR a=150 TO 516
22 LET b=PEEK a
23 IF sw<>0 THEN GO TO 40
24 LET bh=INT(n/16):LET bl=n−bh*16
26 LET b$=h$(bh+1)+h$(bl+1)
28 LET C$=STR$ n +“ ”:LET C$=C$(1 TO 4)
30 LPRINT C$;“ ”; b$; “ ”;:LET SW=1
40 IF b<128 THEN LPRINT CHR$ b;: GO TO 60
55 LPRINT CHR$ (b−128):LET n=n+1:LET SW=0
60 NEXT A
```

Eseguendo il programma, ti appariranno sulla stampante (o sul video, sostituendo gli LPRINT con PRINT) le corrispondenze tra i numeri e le parole riservate.

Gli interrupt

La CPU usa l'area di stack per numerose operazioni, tra le quali anche la gestione degli interrupt.
La comunicazione tra la CPU e le periferiche può avvenire con due tecniche diverse. La

prima prende il nome di "polling" (interrogazione ciclica): la CPU interroga ciclicamente, secondo un ordine prestabilito, i dispositivi esterni, usando un programma che legge lo stato degli stessi. Quando uno dei dispositivi è pronto a trasmettere o a ricevere un dato la CPU manda il programma necessario all'operazione richiesta. La priorità tra i diversi dispositivi è determinata dall'ordine con cui questi ultimi vengono interrogati.
Questa tecnica presenta un solo vantaggio, e cioè di essere realizzata completamente via software, e quindi di non richiedere circuiteria addizionale per le comunicazioni. Presenta però diversi svantaggi, che in alcuni casi la rendono addirittura impraticabile: la CPU è praticamente occupata per la maggior parte del tempo a interrogare dispositivi, mentre solo una minima parte di questo è impiegata per la comunicazione vera e propria. Inoltre il tempo di risposta della CPU alle richieste dei dispositivi può essere in certi casi troppo lungo: se ad esempio i dispositivi sono

parecchi, e uno di essi richiede un'operazione di I/O immediatamente dopo essere stato interrogato, deve aspettare parecchio prima di essere servito; in certi casi questo può comportare la perdita di dati.

La seconda tecnica, che elimina questi svantaggi, è quella degli interrupt. Sono gli stessi dispositivi a segnalare alla CPU la richiesta di un'operazione di I/O, inviando un segnale, (chiamato appunto interrupt, o interruzione) per richiedere alla CPU di interrompere il programma che sta eseguendo, per effettuare l'operazione di I/O. Il servizio di interrupt avviene con un salto a una routine, che prende il nome di routine di servizio dell'interrupt, la quale provvede ad eseguire l'operazione richiesta. Questa tecnica presenta

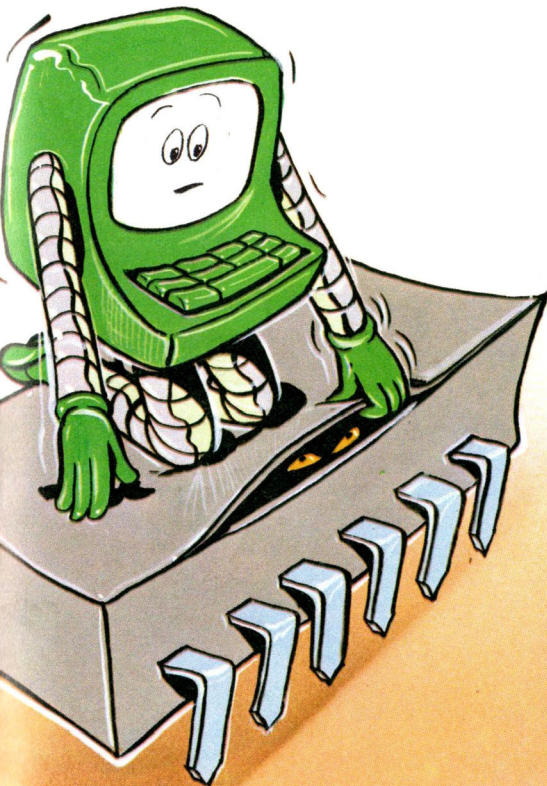
molte analogie con l'esecuzione della subroutine, con la differenza che l'esecuzione della routine di servizio avviene in momenti non prevedibili dal programma, che dipendono dalle esigenze dei dispositivi esterni.

È chiaro che la tecnica degli interrupt elimina gli svantaggi cui si era prima accennato. Il tempo di risposta è infatti limitato solo dalla velocità della CPU per trasferire il controllo da una zona a un'altra della memoria; inoltre l'unità centrale può dedicarsi ad altri programmi, utilizzando in modo più efficiente il suo tempo. La differenza tra polling e interrupt è la stessa che ci sarebbe tra aprire la porta di casa ad intervalli di tempo prefissati, per vedere se c'è qualcuno, e il rispondere al trillo del campanello. La perdita di tempo nel primo caso è evidente, come lo è la scomodità del servizio per chi, desiderando comunicare, deve attendere il successivo controllo per poterlo fare. Dobbiamo però dire che per la gestione delle interruzioni, oltre al dispositivo (il campanello) che segnali la richiesa di comunicazione è necessario predisporre un hardware più complicato. Prima di tutto occorrono una o più linee della CPU dedicate al ricevimento dei segnali di interrupt; in più occorre che lo stesso interrupt si faccia riconoscere dalla CPU, e anche questo rende più complessi i collegamenti; infine, anche se non sempre, è necessaria della circuiteria che consenta il servizio dei diversi dispositivi, in base alla priorità prefissata.
Tuttavia, dal momento che la tecnica dell'interrupt è così vantaggiosa dal punto di vista pratico, quasi tutti i costruttori di computer - escludendo casi particolari - vi ricorrono abitualmente. La possibilità di utilizzo degli interrupt è inoltre estesa anche ai normali programmatori da particolari istruzioni di cui è dotata la CPU. È quindi importante sapere dell'esistenza degli interrupt per due motivi: 1) chiunque li può usare nei propri programmi assembler; 2) il loro funzionamento aiuta a capire tutte quelle azioni che il computer esegue senza che vi sia un intervento diretto della mano dell'uomo.

Usare la ROM

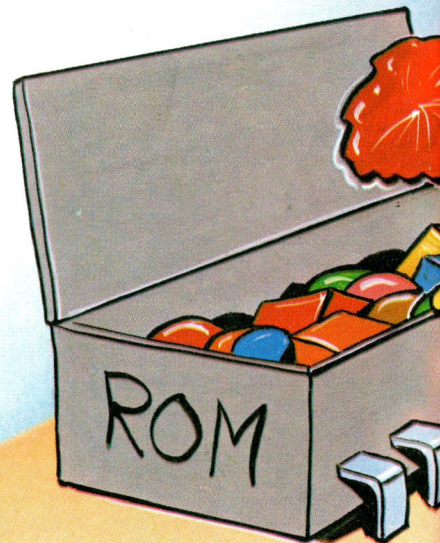
Esistono molte routine del sistema che possono consentire al programmatore esperto di risparmiare il tempo e la fatica di doverle riscrivere. I progettisti di calcolatori strutturano infatti le routine di sistema in maniera tale da poterle considerare come dei normali sottoprogrammi, richiamabili in qualsiasi momento sia da BASIC che da Assembler. Il vantaggio di una simile soluzione è più che evidente: si evita ai programmatori di dover affrontare tutte le volte gli stessi problemi (per esempio la visualizzazione dei risultati), consentendo loro di concentrarsi sul problema specifico piuttosto che sul problema generale.
Inoltre le routine di sistema - poste nella memoria ROM e quindi non cancellabili - sono scritte e controllate da programmatori professionisti, con la conseguente garanzia di sicuro funzionamento.
Per poter utilizzare una qualsiasi di queste routine l'unica cosa che si deve conoscere è l'indirizzo di partenza, oltre naturalmente ai registri del microprocessore che vengono interessati dalla routine stessa. Esistono diversi manuali che descrivono con notevole precisione tutte queste routine; noi tratteremo le principali, spiegando quindi (ed è questo che conta) come utilizzarle nei programmi.
Ciascuna di queste routine possiede un particolare nome mnemonico - normalmente assegnato dalla casa madre - che le permette di essere distinta in modo semplice ed immediato dalle altre. La seguente tabella contiene quindi, oltre all'indirizzo di

partenza, anche il nome di ciascuna routine.

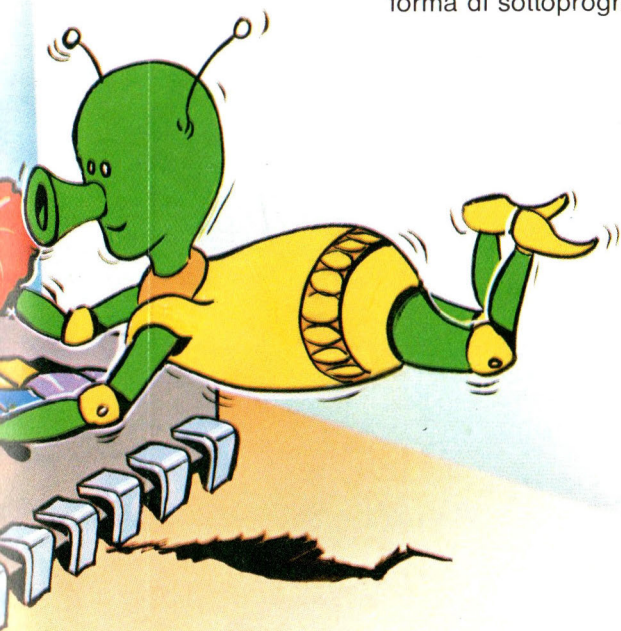
NOME	INDIRIZZO	SCOPO
PRINTOUT	09F4	visualizza sullo schermo
START	0000	inizializza il sistema
KEYBOARD	02BF	controlla la tastiera
BEEPER	03B5	suona l'altoparlante
SAVE	0605	routine di SAVE
LOAD	0808	routine di LOAD
CLS	0D6B	pulisce lo schermo
NEW	11B7	routine di NEW
PLOT	22DC	disegna un punto
SCROLL	0DFE	scrolling di 1 riga

Come esempio di utilizzo di una di queste routine, vediamo in che modo è possibile stampare qualcosa sullo schermo. Dalla tabella appena scritta si deduce che la routine adibita alla visualizzazione dei caratteri sul video è PRINTOUT; essa richiede semplicemente che, prima della chiamata, il codice ASCII del carattere da visualizzare venga posto nell'accumulatore.

Dovremo quindi memorizzare un certo codice nell'accumulatore e chiamare la routine tante volte quanti saranno i caratteri che vogliamo far stampare. Per stampare la parola "ciao", potremo quindi scrivere:

```
LD A,43H        ;ASCII di "C"
CALL 09F4

LD A,49H        ;ASCII di "I"
CALL 09F4

LD A,41H        ;ASCII di "A"
CALL 09F4

LD A,4FH        ;ASCII di "O"
CALL 09F4

LD DE,0105H
LD HL,066AH
CALL 03B5
RET
```

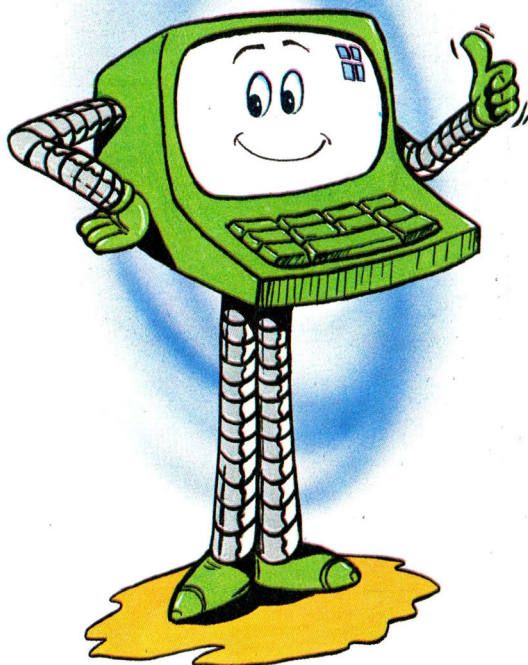
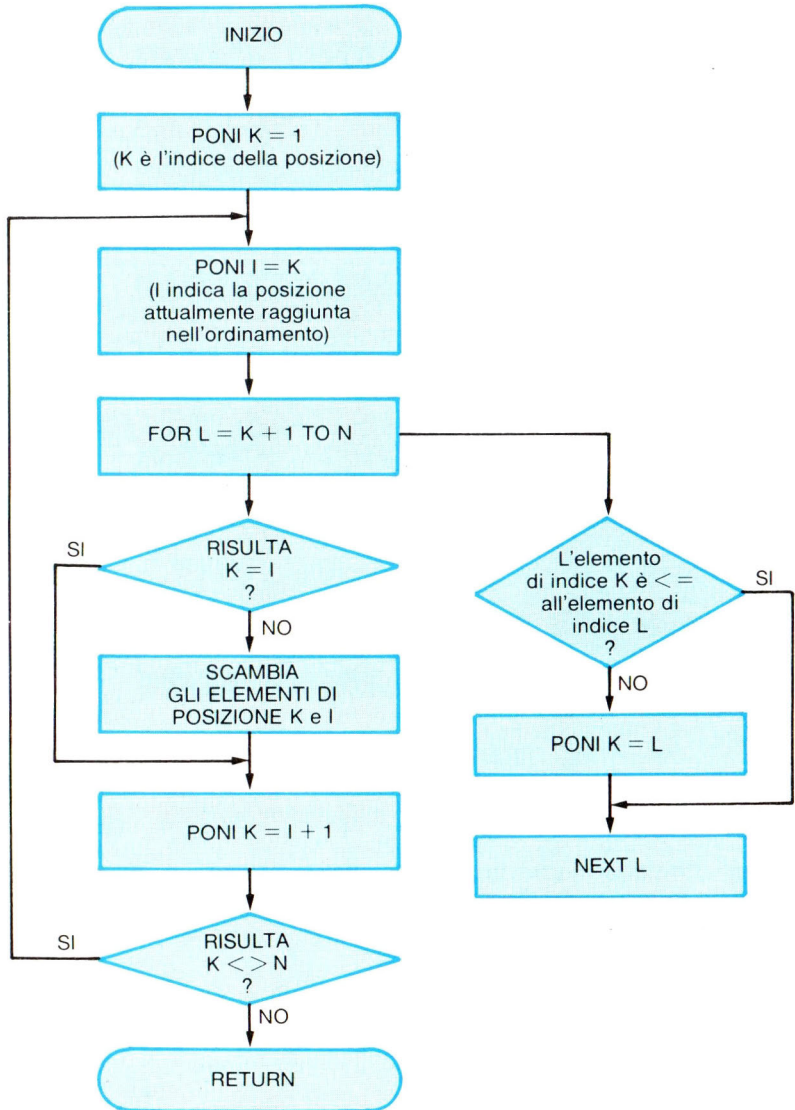
All'operazione di stampa abbiamo anche aggiunto, al termine, una suonatina dell'altoparlante, mediante il ricorso alla subroutine BEEPER. Poiché questa routine richiede che nei registri DE e HL si trovino dei valori numerici corrispondenti alla nota da suonare, prima della CALL, abbiamo impostato tali valori (ed hai anche avuto una

dimostrazione di come non sia possibile usare le routine in ROM semplicemente conoscendone l'indirizzo).
Come puoi vedere, il fatto di conoscere l'esistenza di PRINTOUT ci ha evitato qualsiasi preoccupazione per quanto riguarda l'uscita dei risultati sul video. La chiamata avviene semplicemente attraverso la solita istruzione CALL, a ulteriore dimostrazione che nella ROM le routine si trovano scritte in forma di sottoprogrammi.

La velocità di esecuzione

Ci proponiamo adesso di risolvere il seguente problema: porre in ordine crescente dei valori, disposti a caso, appartenenti a un certo vettore numerico. Desideriamo dunque scrivere un programma di ordinamento. Abbiamo già affrontato questo problema in una delle nostre lezioni; oggi proporremo tuttavia due distinte (ma identiche) soluzioni: la prima in BASIC e la seconda in Assembler. Questo perché tu possa toccare veramente con mano la notevole differenza di velocità tra un linguaggio e l'altro.
La tecnica che utilizzeremo per compiere il lavoro è molto semplice, e può essere riassunta così:
— su un vettore di N numeri operiamo dei confronti ciclici, in tutto N−1 cicli
— per il primo ciclo confrontiamo il primo elemento con tutti gli altri; quando troviamo un numero minore continuiamo il confronto con esso. Alla fine del ciclo portiamo nella prima posizione il

numero più piccolo che abbiamo trovato, scambiandolo con quello che vi ci stava, a meno che sia già esso il minore. Questa operazione si programma facilmente, servendosi degli indici della matrice.

— ogni ciclo mette a posto un elemento, partendo dall'indice minore; di conseguenza ogni ciclo diminuisce di uno il numero degli elementi da confrontare.

— alla fine, dopo aver eseguito N−1 cicli, gli elementi saranno tutti ordinati.

Visto che il cuore del programma è costituito dalla fase di ordinamento, utilizzeremo una subroutine per svolgere questo compito (così facendo potremo infatti utilizzare lo stesso programma principale, sia per la versione in BASIC che per quella in Assembler).

Lo schema a blocchi della routine di ordinamento è il seguente:

Il programma completo è quindi:

```
  10 REM richiesta dati iniziali
  15 CLS : INPUT "Quanti numeri: ";N
  20 IF N>255 THEN GO TO 15
  30 REM dimensiona l'array ed estrae i numeri
  40 DIM V(N)
  50 FOR K=1 TO N
  60 LET V(K)=INT(RND(0)*5000)
  70 NEXT K
  80 GOSUB 500:REM visualizza l'array "disordinato"
  90 GOSUB 1000:REM ordina l'array
 100 GOSUB 500:REM visualizza l'array "ordinato"
 110 STOP
 500 FOR K=1 TO N
 510 PRINT V(K),
 520 NEXT K
 530 PRINT:PRINT
1000 REM Routine di ordinamento
1010 LET K=1
1015 LET I=K
1020 FOR L=K+1 TO N
1030 IF V(K)<=V(L) THEN GO TO 1050
1040 LET K=L
1050 NEXT L
1060 IF K=I THEN GO TO 1080
1070 LET C=V(I):LET V(I)=V(K):LET V(K)=C
1080 LET K=I+1: IF K<>N THEN GO TO 1015
1090 RETURN
```

Vediamo brevemente il funzionamento del programma:
linee 10-20: viene chiesto il numero di elementi da ordinare; se tale numero supera 255 (abbiamo posto come limite questo valore), la richiesta è ripetuta
linee 30-40: l'array viene dimensionato
linee 50-70: vengono estratti gli N numeri da inserire nel vettore
linee 80-100: l'array viene prima visualizzato, poi ordinato e quindi nuovamente visualizzato (per verificare l'avvenuto ordinamento)
linee 500-530: routine di visualizzazione dell'array
linee 1000-1090: routine di ordinamento.
Eseguendo il programma, e imponendo un numero di elementi pari a 255, otterrai l'ordinamento del vettore in un tempo che mediamente (a seconda delle sequenze di numeri che di volta in volta vengono estratti) si aggira sugli 800 secondi (circa 13 minuti).
Ben altri tempi risulteranno invece facendo eseguire il programma con la stessa routine di ordinamento scritta in Assembler. Ecco le linee di programma che dovrai

aggiungere a quelle appena viste, per poter scegliere tra ordinamento BASIC o Assembler.

```
   5 CLEAR 59999
  25 INPUT "Ordinamento BASIC o Assembler (B/A) ";R$
  85 IF R$="A" OR R$="a" THEN GO SUB 2000: GO TO 100
2000 RESTORE 2090
2010 FOR K=60000 TO 60102:READ L:POKE K,L:NEXT K
2020 REM Prepara la chiamata della routine
2030 LET C=PEEK 23627+256*PEEK 23628
2040 LET Y1=PEEK(C+3)-1:POKE 65533, Y1:POKE 65532, Y1
2050 LET Y1=C+41:LET Y2=INT(Y1/256):LET Y3=Y1-Y2*256
2060 POKE 65528,Y3:POKE 65529,Y2
2070 RANDOMIZE USR 60000
2080 RETURN
2090 DATA 42,248,255,229,209,205,193,234,19
2100 DATA 26,35,150,40,6,56,16,229,209,24,12
2110 DATA 43,27,26,150,40,8,56,6,229,209,24,2
2120 DATA 43,27,205,193,234,229,33,252,255,53
2130 DATA 225,40,2,24,217,42,248,255,124,146
2140 DATA 32,6,125,147,32,2,24,16,78,35,70,43
2150 DATA 26,119,35,19,26,119,27,121,1,19,120
2160 DATA 18,42,248,255,205,193,234,34,248,255
2170 DATA 33,253,255,53,32,1,201,126,43,119
2180 DATA 24,159,35,35,35,35,35,201
```

La linea 5 abbassa il TOP della memoria BASIC, per poter caricare la routine in linguaggio macchina. Le linee 25 e 85 servono per poter scegliere l'ordinamento che si desidera (BASIC o Assembler).
Le linee 2000-2180 inseriscono invece in memoria i codici della routine, inizializzando e preparando alcune

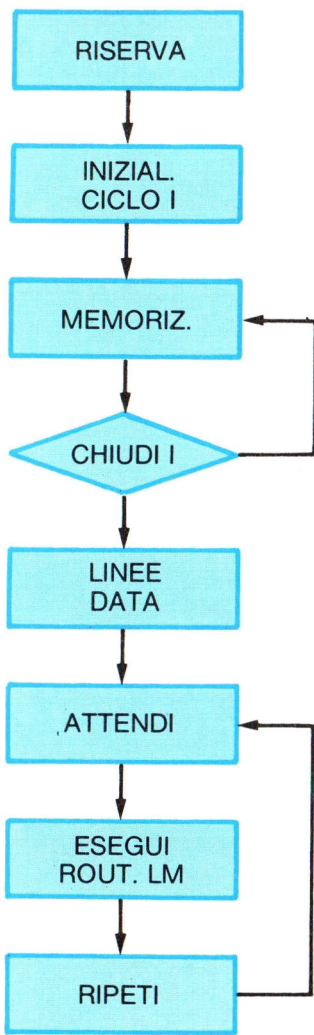
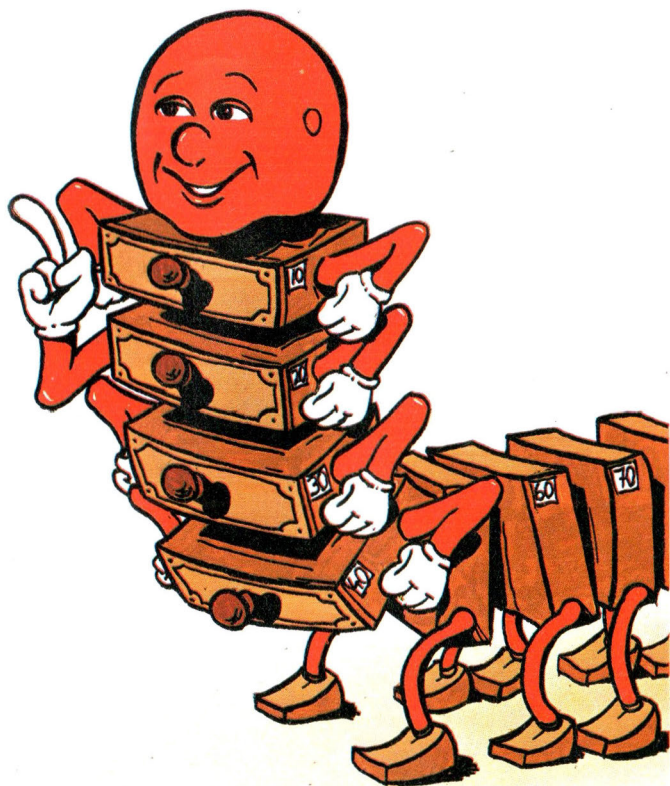
locazioni di memoria necessarie al funzionamento del programma Assembler. In particolare la linea 2070 fa partire l'esecuzione della routine.

Il tempo medio di ordinamento ottenibile lavorando con il programma in linguaggio macchina è di circa 2 secondi: in questo caso (anche se non sempre accade cosi) il vantaggio dell'Assembler rispetto al BASIC è di 1 a 400!

Print senza PRINT

È un buon esercizio ed una valida dimostrazione di come utilizzare il sistema operativo in ROM scavalcando l'interprete BASIC. Ecco la parte LM.

LD A, 2	Prepara il rag A per ...
CALL 1601	... attivare il canale del display
LD A, (23560)	Immetti LASTK (ultimo tasto)
RST 10	Chiama la routine di stampa
RET	Ritorna

```
10 CLEAR 29999
20 FOR I = 30000 TO 30009
30 READ X
40 POKE I, X
50 NEXT I
60 DATA 62, 2, 206, 1, 22, 58, 8, 92, 215, 201
70 PAUSE 0
80 RANDOMIZE USR 30000
90 GO TO 70 LM : 30000 X : VAR. COMODO
```

VIDEOESERCIZI

Quali sono i più veloci tra i due seguenti programmi?
Cronometro alla mano cerca di scoprirlo e di spiegartene il motivo. Ricorda però che la velocità può essere importante, a volte essenziale ma che la leggibilità e la chiarezza del listato lo sono ancora di più.

```
10 FOR P = 1 TO 1000
20 REM Questo commento
rallenta; a meno
che non sia indispen-
sabile è meglio ometterlo
30 NEXT P
```

```
10 FOR P = 1 TO 1000 :
NEXT P
```

```
10 POKE 23692, 255
20 LET N$ = "23"
30 FOR I = 1 TO 1000
40 PRINT N$;
50 NEXT I
```

```
10 POKE 23692, 255
20 LET N = 23
30 FOR I = 1 TO 1000
40 PRINT N;
50 NEXT I
```